STOCKHOLM UNIVERSITY
Department of Statistics
Ellinor Fackle Fornius

Hand-in assignment 3

R Programming

This document contains R course material designed by Leif Johnsson, Måns Magnusson and Josef Wilzén

Contents

Instructions

The assignments have to be written in `R markdown` with the output in either `pdf` or `HTML` format. Both the `.Rmd` and `.pdf`/`.html`-files have to be handed in. Upload the documents to Athena no later than the deadline for this assignment (stated in the Course Description). Name your files with your group ID and assignment number. The report should include the code as well as the corresponding output. Comment your code when appropriate. For each task in the assignment you should reproduce the example code, when given. Otherwise, construct a relevant example of your own to illustrate the results.

1 `my_matrix_prod()`

A central part of linear algebra is matrix multiplication, i.e. to multiply two matrices by each other. You will now create a function called `my_matrix_prod()` with the arguments A and B multiplying two matrices by each other in the following way:

$$\texttt{my\_matrix\_prod(A, B)} = \mathbf{A} \cdot \mathbf{B}$$

If the dimensions do not make it possible to multiply the matrices the function should **stop** and return the error message `Matrix dimensions mismatch`.

Say we want to calculate $C = A \cdot B$. Matrix multiplication is defined so that elements $C[i, j] = A[i,] \cdot B[, j]$ if we use R's notation for indexing. If we want to calculate the element on row 3 and column 4 in C we take the scalar product between the vectors a_3 and b_4. Where a_3 is the vector which consists of row 3 in the matrix A and b_4 is the vector which consists of column 4 of the matrix B. To get all elements in C we must do this procedure for all combinations of rows from A and columns from B.

Note that it is not allowed to use R's function for matrix multiplication `%*%`. However, you may use it to generate more test cases to test that your function counts correctly.

Possible steps for the function are as follows:

1. Test if the dimensions of the matrix **A** och **B** means that they can be multiplied by each other, otherwise stop the function and return the error message (see below).
2. Create a new matrix (eg called **C**) with the dimensions which the product of A and B has.
3. Loop over the elements in **C** and calculate each element separately. This can be done with a nested for-loop, where one loop goes over rows and the other goes over columns. **Hint!** Here you can use your code from **orth_scalar_prod()** in Assignment 1.

Here are text examples of how the function should work:

```
X <- matrix(1:6, nrow = 2, ncol = 3)
Y <- matrix(6:1, nrow = 3, ncol = 2)
my_matrix_prod(A = X, B = Y)

     [,1] [,2]
[1,]   41   14
[2,]   56   20

test_mat <-my_matrix_prod(A = Y, B = X)
test_mat
```

```
     [,1] [,2] [,3]
[1,]   12   30   48
[2,]    9   23   37
[3,]    6   16   26

my_matrix_prod(A = X, B = X)

Error in my_matrix_prod(A = X, B = X): Matrix dimensions mismatch
```

2 sum_of_random_dice()

Function `sum_of_dice()` in Computer lab 5 sums the value from a fixed number of dices. Now you have to write one function that can sum up a random number of dices. The number of dices must be Poisson distributed with a parameter λ in the following way.

$$\begin{aligned} Y &\sim \text{Po}(\lambda) \\ X &= \sum_{i}^{Y} Z_i \end{aligned}$$

where Z_i is the outcome of a six-sided dice. Create a function `sum_of_random_dice()`. The arguments should be:

- `K`: number of draws from the random distribution
- `lambda`: a positive continuous number (parameter in the Poisson distribution)
- `my_seed`: a random seed that controls the random number generation, default shall be `NULL`.

The function `sum_of_random_dice()` should return a `data.frame` with the sum of the eyes on the random number of dices as well as the number of dices thrown.

A suggestion on how this can be implemented can be found here:

- Change the seed to: `set.seed(my_seed)`.
- Set up a blank `data.frame` named `result`, which should have `K` rows and 2 columns. The first column should have the name `value` and the other `dice`.
- Do the following for-loop over the vector `1:K`
  - Subtract a random number from a Poisson distribution with parameter `lambda`. Save the random number in `current_number`, which is the number of dices. Save `current_number` in the column `dice` in the correct row in `result`.
  - Call `sum_of_dice()` with arguments `K=1` and `N=current_number`, save the result in column `value` in the correct row in `result`. Note that if `curret_number=0` the sum should be 0 .
- Return `result`.

Note! Even if you do not implement the function in this way, it can still work properly. But because the random numbers are used in different order, it may be that you do not get the same results as in the

examples below. The results may also be different if another version of R is used. However, make sure that your results are reasonable and that you get the same result when you reuse a sample seed.

Test if the test cases below work:

```
sum_of_random_dice(K = 5, lambda = 3, my_seed = 42)
```

```
  value dice
1    13    5
2     8    4
3    18    6
4    26    6
5    10    4
```

```
sum_of_random_dice(K = 5, lambda = 8, my_seed = 4711)
```

```
  value dice
1    47   13
2    21    5
3    30    8
4    28    9
5    20    8
```

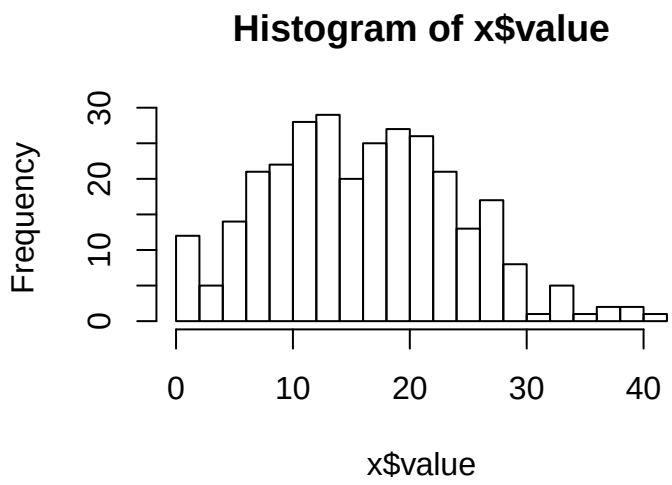
```
# No seed give different results
sum_of_random_dice(K = 5, lambda = 3)
```

```
  value dice
1    10    2
2    16    5
3     4    1
4    18    5
5     3    2
```

```
sum_of_random_dice(K = 5, lambda = 3)
```

```
  value dice
1    14    5
2     9    3
3    23    5
4    10    4
5    13    4
```

```
x <- sum_of_random_dice(K = 300, lambda = 5, my_seed = 42)
hist(x$value, 20)
```



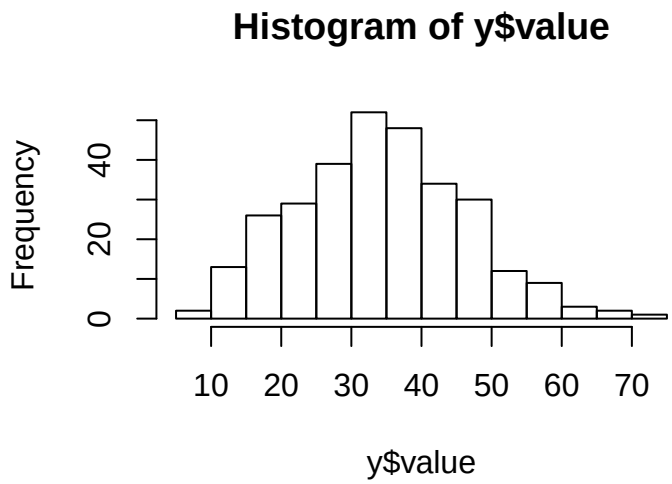

```
mean(x$value)

[1] 16.61

sd(x$value)

[1] 8.1899

y <- sum_of_random_dice(K=300, lambda=10, my_seed=4711)
hist(y$value, 20)
```



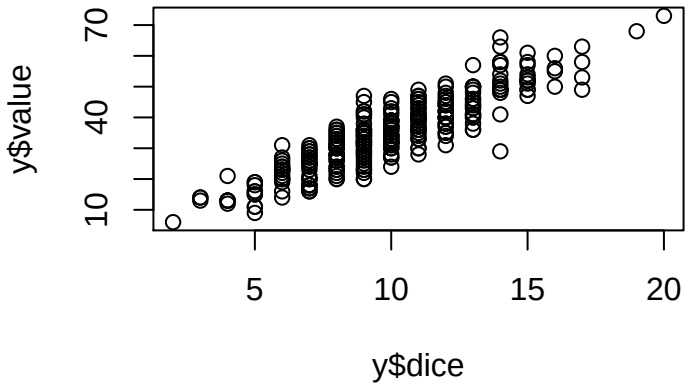

```
mean(y$value)

[1] 34.72

sd(y$value)

[1] 12.017

plot(y$dice, y$value)
```



3 `my_ols()`

In this problem you should write a function that can estimate a linear regression model. Linear regression models a linear relationship between a dependent variable (y) and one or more independent variables ($x_1, x_2, x_3, \ldots$). There is a built-in function in R for linear regression, `lm()`, but it is not allowed in this task. Instead you have to "estimate the model by hand" using matrix algebra and the least squares method.

Linear regression is about investigating if a dependent variable y depends on one or several independent/explanatory variables $x_1, x_2, ..., x_p$. These column vectors are usually put together into a matrix $\mathbf{X}$.

A linear regression model with one independent variable looks like this:

$$y = \beta_0 + \beta_1 \cdot x_1 + \epsilon$$

where $\epsilon \sim N(0,\ \sigma_\epsilon^2)$.

This is the equation of the straight line in an x-y plane. If we have several, e.g. three independent variables, the model looks like:

$$y = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \beta_3 \cdot x_3 + \epsilon$$

where $\epsilon \sim N(0,\ \sigma_\epsilon^2)$.

If we are to estimate a linear regression, it means that we must find the best values for the parameters in some sense, which is the task here. When we have estimated the parameters β and σ_ϵ^2 from the data, we use the terms $\hat{\beta}$ and $\hat{\sigma}_\epsilon^2$. Once we have the parameter estimates we can calculate expected values $\hat{y}$:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot x_1 + \hat{\beta}_2 \cdot x_2 + \hat{\beta}_2 \cdot x_2 + \hat{\beta}_3 \cdot x_3$$

The residuals are defined as:

$$\hat{\epsilon} = y - \hat{y}$$

We will now create the function `my_ols(X, y)` which has the arguments:

- `X`: an array where the columns are the independent variables in our model

- `y`: a vector with the variable that is the dependent variable in the model.

The function should return a list that contains $\hat{\beta}$, $\hat{\sigma}_\epsilon^2$ and the residuals. `Note!` You may not use the function `lm()` in this task.

The solution sequence follows below, the dataset `attitude` in R is used as an example. In this problem we will calculate $\hat{\beta}$ and $\hat{\sigma}_\epsilon^2$ as well as the residuals for the model. For these calculations, the least square method is used, which gives:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$$\hat{\sigma}_e^2 = \frac{(\mathbf{y} - \mathbf{X}\hat{\beta})^T(\mathbf{y} - \mathbf{X}\hat{\beta})}{n - p}$$

Start from the data material `attitude` and read it in R on the following way.

```
data(attitude)
```

The following steps describes a way to implement the least squares method in R.

1. The following linear regression model

$$\texttt{rating} = \beta_0 + \beta_1 \cdot \texttt{complaints} + \beta_2 \cdot \texttt{privileges} + \beta_3 \cdot \texttt{learning}$$

can be estimated by matrix algebra. Perform the following steps to make this calculations. Then implement this as a general function.

   (a) Prepare $\mathbf{X}$

      i. Convert your X to a matrix if it is a data.frame and rename it matrix to `X`. **Hint!** `as.matrix()`

      ii. Add a column vector with ones as the first column in $\mathbf{X}$ (this is to our intercept, β_0). Name this column "(Intercept)". See example below.

```
     (Intercept) complaints privileges learning
[1,]           1         51         30       39
[2,]           1         64         51       54
[3,]           1         70         68       69
[4,]           1         63         45       47
[5,]           1         78         56       66
[6,]           1         55         49       44
```

   (b) Prepare $\mathbf{y}$

      i. Turn your vector y into a $n \times 1$ matrix.

```
     [,1]
[1,]   43
[2,]   63
[3,]   71
[4,]   61
[5,]   81
[6,]   43
```

   (c) Transpose ($\mathbf{X}^T$) your matrix `X`.

(d) Calculate the matrix multiplication $\mathbf{X}^T\mathbf{X}$.

(e) Calculate $(\mathbf{X}^T\mathbf{X})^{-1}$

(f) Calculate your estimate of $\beta_0, \beta_1, \beta_2, \beta_3$ as follows: $\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ and save the result as a 4×1 matrix and name it `beta_hat`. Check that the rows in `beta_hat` have the same name as the variables i `X`. Calculate the expected value (prediction) $\widehat{\mathbf{y}}$ for any observation, i.e.

$$\widehat{\mathbf{y}} = \mathbf{X}\hat{\beta}$$

Calculate the residuals $\hat{\epsilon} = y - \hat{y}$ and name them `e_hat`. **Hint!** Check that `e_hat` is a $n \times 1$ matrix and not a vector, otherwise the function `as.matrix()` can be used.

(g) Calculate how many observations were used in the analysis (number rows in $\mathbf{X}$), save this value as `n`.

(h) Calculate how many parameters are estimated in the model, ie how many columns is it in $\mathbf{X}$, save this value as `p`.

(i) Calculate

$$\hat{\sigma}_e^2 = \frac{\hat{\epsilon}^T\hat{\epsilon}}{n-p}$$

and save $\hat{\sigma}_e^2$ as `sigma2_hat`.

(j) If you did (a) - (k) above and everything works well, then create a function that make these calculations. Save `beta_hat`, `sigma2_hat` and `e_hat` in a list with the list names `beta_hat`, `sigma2_hat` and `e_hat`. The order should be as above.

(k) Set the class for the list to `my_ols`.

(l) Return the list.

Now check if your function can handle the following test cases:

```
data(attitude)
X <- attitude[, 2:4]
y <- attitude[, 1]
inherits(my_ols(X, y), "my_ols")
```

```
[1] TRUE
```

```
class(my_ols(X, y))
```

```
[1] "my_ols"
```

```
my_ols(X, y)[1:2]
```

```
$beta_hat
                 [,1]
(Intercept) 11.25831
complaints   0.68242
privileges  -0.10328
learning     0.23798

$sigma2_hat
       [,1]
[1,] 47.101
```

```
head(my_ols(X, y)[["e_hat"]])
```

```
          [,1]
[1,]  -9.24409
[2,]   0.48382
[3,]   2.57551
[4,]   0.21236
[5,]   6.59070
[6,] -11.20124
```

```
data(trees)
trees_ols <- my_ols(X = trees[, 1:2], y = trees[, 3])
trees_ols[1:2]
```

```
$beta_hat
                 [,1]
(Intercept) -57.98766
Girth         4.70816
Height        0.33925

$sigma2_hat
       [,1]
[1,] 15.069
```

```
summary(trees_ols[[3]])
```

```
       V1
 Min.   :-6.407
 1st Qu.:-2.649
 Median :-0.288
 Mean   : 0.000
 3rd Qu.: 2.200
 Max.   : 8.485
```

```
A <- my_ols(X, y)

names(A)
```

```
[1] "beta_hat"   "sigma2_hat" "e_hat"
```

```
X2 <- attitude[, 2, drop = FALSE]
B <- my_ols(X = X2, y = y)

str(A)
```

```
List of 3
 $ beta_hat  : num [1:4, 1] 11.258 0.682 -0.103 0.238
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:4] "(Intercept)" "complaints" "privileges" "learning"
  .. ..$ : NULL
```

```
 $ sigma2_hat: num [1, 1] 47.1
 $ e_hat     : num [1:30, 1] -9.244 0.484 2.576 0.212 6.591 ...
 - attr(*, "class")= chr "my_ols"

str(B)

List of 3
 $ beta_hat  : num [1:2, 1] 14.376 0.755
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:2] "(Intercept)" "complaints"
  .. ..$ : NULL
 $ sigma2_hat: num [1, 1] 48.9
 $ e_hat     : num [1:30, 1] -9.861 0.329 3.801 -0.917 7.764 ...
 - attr(*, "class")= chr "my_ols"
```

4 my_grouped_test()

Now you will create a function that will calculate group-by-group confidence intervals (KI) for a variable. Before you start make sure that the HUS data is loaded and run the code below to remove the extremely large values:

```
# Download

HUS <- read.csv(file = 'HUS_eng.csv')

# Small corrections (removing outliers)
index <- HUS[, 1] < quantile(HUS[, 1])[4]

HUS <- HUS[index,]
```

The function should be called `my_grouped_test()` and have the arguments:

- `data_vector` - is a numeric vector
- `my_groups` - is a factor/character vector that groups `data_vector`
- `alpha` - is the significance level for the range. `alpha=0.05` shall provide a 95 % confidence interval.

The function should return a matrix `result` where the rows corresponds to groups in `my_groups` and has four columns: Lower limit for KI, the mean, upper limit for KI and number of observations in each group. See the test cases for the names of the columns. The rows shall have the same names as the groups `my_groups`.

Suggested solution:

1. Make sure `my_groups` is a factor. Calculate how many groups can be found in `my_groups`. **Hint!** `?levels()` `?table()`
2. Create an empty matrix of the correct size, call it `result`. Give it appropriate names. **Hint!** `?colnames()`, `?rownames()`

3. Save the number of observations for each group in the fourth variable in `result`.
4. Use `by()` combined with `t.test()` to calculate groupwise KI, save in `group_test`. Test `?by()`, you see that there is an argument called "`...`" for function `by()`. These three points can be replaced by arguments needed for the "`FUN`" function. More tips: `str("object from t.test")` and `?by`, read under the heading "value" to check what `by()` returns.
5. Loop over the number of groups and select CI and mean for each group from `group_test`. Save in the correct places in `result`.
6. Return `result`.

Test if the test cases below work:

```
my_grouped_test(HUS[, 1], HUS$air.conditioning,0.01)

  Lower CI-limit   Mean Upper CI-limit No of obs.
0         161468 173473         185479         82
1         213182 220556         227930        308

my_grouped_test(HUS[,2], HUS$pool, 0.10)

  Lower CI-limit   Mean Upper CI-limit No of obs.
0         1918.7 1955.5         1992.3        372
1         1890.6 2041.5         2192.4         18

# Chickwts data
data(chickwts)
my_grouped_test(chickwts[, 1], chickwts[, 2], 0.05)

          Lower CI-limit   Mean Upper CI-limit No of obs.
casein            282.64 323.58         364.52         12
horsebean         132.57 160.20         187.83         10
linseed           185.56 218.75         251.94         12
meatmeal          233.31 276.91         320.51         11
soybean           215.18 246.43         277.68         14
sunflower         297.89 328.92         359.95         12
```

5 give_blood()

For blood donors, there are certain rules for when they can donate blood. Here you can find the rules which apply. You will write a function that should help a blood donor know when to donate blood, based on some of the rules. The function should be called `give_blood()` and have the arguments:

- `lasttime`: a date that indicates the last time the blood donor gave blood, default should be today. **Hint!** `today()`
- `holiday`: should be either: 1) an interval object indicating start and end dates for a trip abroad. Start date is the date the person leaves Sweden and the end date is the date the person arrives back in Sweden. 2) The default value should be "`hemma`", which indicates that there will be no trip.
- `sex`: assumes the value "f" for female and "m" for male

- `type_of_travel`: "`malaria`" indicates travel to a country where there is malaria and "`other`" indicates travel to a country without malaria. Should be NULL if `holiday` has the value "at home".

All dates must be in the form **"year-month-day"**. The function should, given the arguments, calculate a date when the blood donor is allowed to donate blood again and return the date. We assume that the blood donor wants to give blood as **often** as possible. The function must follow the following rules:

- Minimum time between blood transfusions: women 4 months, men 3 months, both indicate relative time. After exactly this time, the person can give blood.
- If the person has been in a country where there is no malaria, he should wait (be in quarantine) 4 weeks (relative time) after the end date of the argument `holiday` before he is allowed to donate blood.
- If the person has been in a country where there is malaria, he should wait (be in quarantine) 6 months (relative time) after the end date of the argument `holiday` before he is allowed to donate blood.
- In the case of quarantine, the person must not donate blood during the quarantine. It will be on the first day after the quarantine the person is allowed to donate blood.
- We assume that the blood donation center is only open on weekdays (Monday to Friday). Given the previous rules, the first possible weekday is chosen.

Below is a proposal for an order of solution:

1. Examine whether the person has been at home, traveling ashore with malaria or in land without malaria. Add any additional time to the end date and save as `extraTime`. Remember to take into account the case when the person does not travel, for example by setting `extraTime` to the same date as `lasttime`. **Hint!** `int_end(), months(), weeks()`
2. Given whether it is a man or woman figure out when the person at the earliest may donate blood, save that date in the variable `suggestion`. **Hint:** `months()` Check if `suggestion` occurs after `extraTime`, if so the case set `suggestion` as a suggestion for blood donation. If not, enter the day after `extraTime` as a suggestion.
3. Check that the specified day is a weekday, if not enter the next weekday as a suggestion. **HInt!** `?wday(), ?days()`
4. Return the proposal as a text string on the form: `"year=[year], month=[month], day=[day], weekday=[weekday]"`.

   For example, if the proposed date is 19/02/2014, the string should be:

   `"year=2014, month=Feb, day=19, weekday=Friday"`. **Hint!** `year(), month(), day(), paste()`

It may be that weekday() returns the weekdays in Swedish. To return weekdays in English, the following settings are available:

```
Sys.setlocale("LC_TIME", "English")
# or
Sys.setlocale("LC_TIME", "C")
```

If you have difficulty changing the language (locale), try the following:

```
# month:
library(lubridate) x <- ymd("2012-03-28")
print(month (x))
print(month.name)
y <- substr(x = month.name[month(x)], start = 1, stop = 3)
print(y)
#weekend:
week_name <- c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")
print(wday(x))
week_name[wday(x)]
```

Check if the function meets the test cases below:

```
library(lubridate)


Attaching package: 'lubridate'
The following object is masked _by_ '.GlobalEnv':
       leap_year
The following objects are masked from 'package:base':
       date, intersect, setdiff, union
```

```
Sys.setlocale("LC_TIME", "C")

[1] "C"

# Test 1:
day1<-ymd("2014-02-24")
print(day1)

[1] "2014-02-24"

give_blood(lasttime=day1, holiday="hemma", sex="m", type_of_travel=NULL)

[1] "year=2014 month=May day=26 weekday=Monday"

give_blood(lasttime=day1, holiday="hemma", sex="f", type_of_travel=NULL)

[1] "year=2014 month=Jun day=24 weekday=Tuesday"

# Test 2:
day2 <- ymd("2014-03-23")
day3 <- ymd("2014-04-24")
holiday1 <- interval(day2, day3)
give_blood(lasttime=day1, holiday=holiday1, sex="m", type_of_travel="malaria")

[1] "year=2014 month=Oct day=27 weekday=Monday"

give_blood(lasttime=day1, holiday=holiday1, sex="f", type_of_travel="malaria")

[1] "year=2014 month=Oct day=27 weekday=Monday"
```

```
# Test 3:
day4 <- ymd("2014-04-13")
day5 <- ymd("2014-05-23")
holiday2 <- interval(day4, day5)
give_blood(lasttime=day1, holiday=holiday2, sex="m", type_of_travel="other")

[1] "year=2014 month=Jun day=23 weekday=Monday"

give_blood(lasttime=day1, holiday=holiday2, sex="f", type_of_travel="other")

[1] "year=2014 month=Jun day=24 weekday=Tuesday"
```

6 Swedish social security number

In Sweden, all citizens have social security numbers that they keep for life and use for identification. The social security number consists of three parts, date of birth, birth number and a control digit. By default social security number are entered as follows `YYYYMMDDNNNK` where `YYYY` is the year of birth, `MM` the month of birth, `DD` the birthday, `NNN` the birth number and `K` the control digit.

The control digit is calculated based on the other digits in the social security number which makes it possible to check whether a social security number is correct or not. It is also possible to, based on a social security number, calculate age and gender (and for some even place of birth, but it plays no role in this task).

The details of how gender and the control digits are calculated can be found in the Statistics Sweden brochure [link]. Read chapter 1. ”Social security number” in this brochure before you do the task below.

Examples of social security numbers that can be used to test your functions can be found partly in the brochure from Statistics Sweden and partly on Wikipedia (keywords: “Social security number in Sweden”). You can of course also test with your own social security number if you want.

We take it in several steps, with different functions that perform different tasks. The steps we will take are:

1. Create a function to check the controldigit in a (arbitrary) social security number.
2. Create a function to retrieve gender information from a (arbitrary) social security number.

Hint! `str_c()/paste()`, `str_sub/substr()` and `Sys.Date()`. Check out these features before you get started.

This functionality for social security numbers are already available for Swedish social security numbers in the package `sweidnumbr`. However, this package must not be used.

Here is a test example of how the function should work:

```
pnr <- "196408233234"
pnr_ctrl (pnr)

[1] TRUE

pnr <- "190101010101"
pnr_ctrl (pnr)

[1] FALSE
```

```
pnr <- "198112189876"
pnr_ctrl (pnr)

[1] TRUE

pnr <- "190303030303"
pnr_ctrl (pnr)

[1] FALSE
```

6.1 pnr_ctrl()

You will now create a function that can check whether a social security number is correct or not. To calculate a control digit the so-called Luhn algorithm is used, more information can be found **here**. Assume that the social security numbers comes in the format `YYYYMMDDNNNK` as a text vector.

The function should take the argument `pnr` and return `TRUE` or `FALSE` depending on whether the social security number is correct or not.

A suggestion on how the function can be implemented is the following:

1. Divide the social security number so that each number becomes a separate element.

   Hint! `str_split()/strsplit()` and `unlist()`

2. Convert the divided numbers to a numerical format.

3. The vector with the individual digits in the social security number can now be used with the Luhn algorithm. The easiest way is to multiply the social security number vector with a calculation vector of 0s, 1s and 2s in the way that the calculation is specified by the Luhn algorithm. **Note!** Calculation is not to be made on the entire social security number, the elements which which will not be utilized can be set to 0 in the calculation vector.

4. The next step is to sum all the values in the vector above. Keep in mind that numbers greater than 9 shall be counted as the sum of the tens digit and the singular digit. **Hint!** `%%` and `%/%`

5. Sum the values of the vector calculated in 4 above. Pick out the singular number and subtract this singular digit from 10. You have now calculated the control digit!

6. Test if the calculated control digit is the same as the control digit in the social security number.

Here is a test example of how the function should work:

```
pnr <- "196408233234"
pnr_ctrl(pnr)

[1] TRUE

pnr <- "190101010101"
pnr_ctrl(pnr)

[1] FALSE

pnr <- "198112189876"
pnr_ctrl(pnr)
```

```
[1] TRUE

pnr <- "190303030303"
pnr_ctrl(pnr)

[1] FALSE
```

6.2 pnr_sex()

In this task, we will calculate the legal sex from a social security number. As stated in the tax brochure, this can be calculated by checking if the penultimate digit of the social security number is even (female) or odd (Man). This is what defines a person's legal sex.

Now create a function you call `pnr_sex()` with the argument `pnr`. This function should take a social security number and return a person's gender as a text element, `M` for man and `K` for woman.

A suggestion on how the function can be implemented is the following:

1. Pick out the penultimate digit of the social security number.
2. Convert this number to numerical format and test if the number is even (return `K`) or odd (return `M`)

Here are test examples of how the function should work:

```
pnr <- "196408233234"
pnr_sex(pnr)

[1] "M"

pnr <- "190202020202"
pnr_sex(pnr)

[1] "K"
```